Introduction
Introduction module in the Sparse Signal Recovery collection.

Introduction

As we progress into the era of information overload, it becomes increasingly important to find ways to extract information efficiently from data sets. One of the key concerns in signal processing is the accurate decoding and interpretation of an input in a minimal period of time. In the distant past, the information theorists' objective primarily involved reducing the signal elements to be processed. Now, with a multitude of extraction options, we are also concerned about the computation time required to interpret each element.

In this project, we investigate a few methods by which we can “accurately” reconstruct an arbitrarily complex signal using a minimum number of iterations. The input signals we probe are deemed **sparse** – that is, they are constructed using a number of basis vectors that is small relative to the length of the signal. This generally means that the signals consist mostly of zero values, with spikes at a few selected positions. We compare the signal-to-noise ratios achieved by our recovery methods after specified numbers of iterations upon a variety of input signals, and deduce a few conclusions about the most efficient and most feasible recovery methods.

Theory
The Theory module within the Sparse Signal Reconstruction in the Presence of Noise collection.

Theory

Motivation

In theory, we should never have to 'recover' a signal – it should merely pass from one location to another, undisturbed. However, all real-world signals pass through the infamous “channel” – a path between the transmitter and the receiver that includes a variety of hazards, including **attenuation**, **phase shift**, and, perhaps most insidiously, **noise**. Nonetheless, we depend upon precise signal transmission daily – in our watches, computer networks, and advanced defense systems. Therefore, the field of signal processing concerns itself not only with the deployment of a signal, but also with its recovery in the most efficient and most accurate manner.

Types of Noise

Noise takes many forms. The various 'colors' of noise are used to refer to the different power spectral density curves that types of noise exhibit. For example, the power density of pink noise falls off at 10dB per decade. The power density spectrum of pink noise is flat in logarithmic space. The most common type of noise, however, is **white noise**. White noise exhibits a flat power density spectrum in linear space. In many physical process (and in this report), we deal primarily with Additive White Gaussian Noise – abbreviated **AWGN**. As a reminder, the Gaussian distribution has the following PDF (Probability Density Function):

$$\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

μ is the mean; $\sigma^2 \geq 0$ is the variance.

Sparse Signals

An additional constraint we imposed upon our input signals was that they were required to be **sparse**. A signal that is sparse in a given basis can be reconstructed using a small number of the basis vectors in that basis. In the standard basis for R^n, for example, the signal (1,0,0,0,...,0) would be as sparse as possible – it requires only the basis vector e_1 for reconstruction (in fact, e_1 is the signal!). By assuming that the original signals are sparse, we are able to employ novel recovery methods and minimize computation time.

Typical Reconstruction Approaches

We have a number of choices for the recovery of sparse signals. As a first idea, we could "**optimally select**" the samples we use for our calculations from the signal. However, this is a complicated and not always fruitful process.

Another approach is **Orthogonal Matching Pursuit (OMP)**. OMP essentially involves projecting a length-n signal into the space determined by the span of a k-component "nearly orthonormal" basis (a random array of 1/sqrt(n) and (-1)/sqrt(n) values). Such a projection is termed a **Random Fourier Projection**. Entries in the projection that do not reach a certain threshold are assigned a value of zero. This computation is iterated and the result obtained is an approximation of the original sparse signal. Unfortunately, OMP itself can be fairly complicated, as the optimal basis is often a wavelet basis. Wavelets are frequency "packets" - that is, localized in both time and frequency; in contrast, the Fourier transform is only localized in frequency.

Signal Reconstruction: Our Method

The fundamental principle for our method of signal analysis is determining where the signal is not, rather than finding where it is. This information is stored in a **mask** that, when multiplied with the **running average** of the signal, will provide the current approximation of the signal. This mask is built up by determining whether a given value in the signal is above a threshold, which is determined by the standard deviation of the noise; if so, the value is most likely a signal element. This process is repeated until the signal expected is approximately equal to a signal stored in a library on the device. While this operation is naturally more noticeable at each iteration with sparse signals, even for non-sparse signals the only limiting factor is the **minimum value** of the signal. For reasons of application, the primary limiting factor is the number of samples required to recover the signal. This is because the raw mathematical operations take fractions of a second to a few seconds to execute (which is more than enough for conventional applications). The signal itself may be transmitted for a very short period; the requisite **number of samples** must be garnered before transmission halts. Further, given an arbitrary amount of computation time, our algorithm can reconstruct a sparse signal contaminated with *any* level of AWGN – there is no mathematical limit on the recovery process. This is an impressive and surprising feat.

Implementation
Algorithm and implementation of arbitrary sparse reconstruction in the Sparse Signal Recovery in the Presence of Noise collection.

Implementation

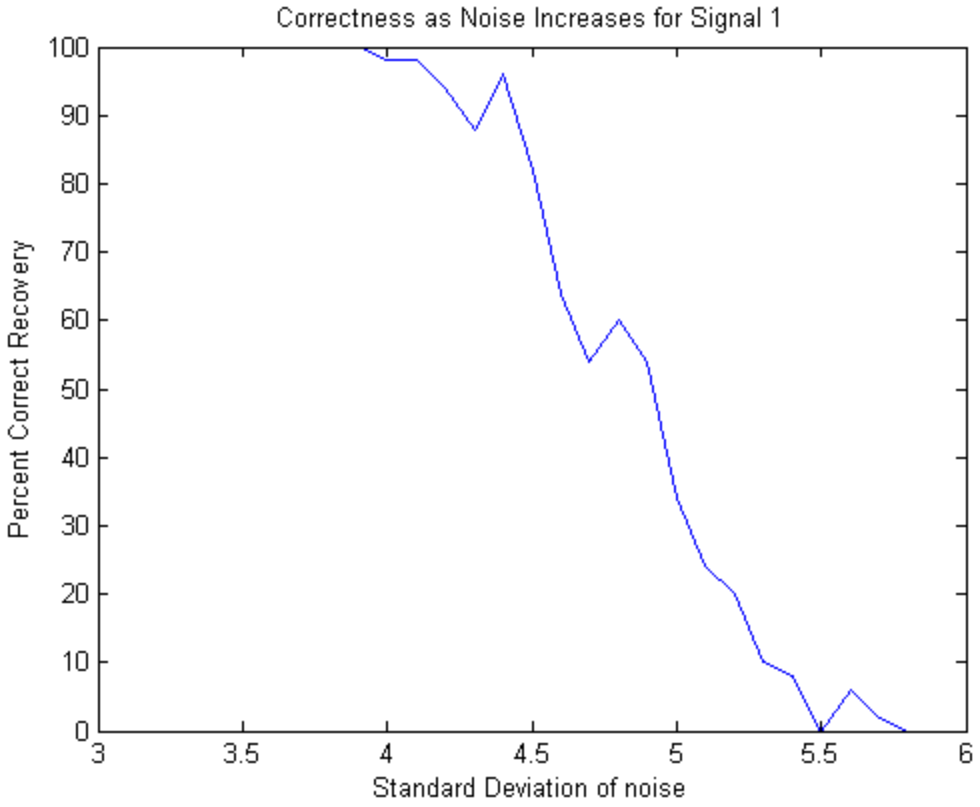
Our implementation of the system was based around a controller program, which accepts the password to be transmitted, and then simulate the transmission and reconstruction. The program then returns whether or not the password received is the same as that which activated the system. In the final application, the controller is called repeatedly until the system is activated; it returns immediately after a non-match, and continues in sequence after the first element is matched.

For each of the passwords element's iterations in the controller the following algorithm is used. The threshold is set to [the minimum value of the ideal signal minus three times the standard deviation of the base noise] (in our case one), and the mask is initialized to all ones. Then we prime the noise to reach either that standard deviation or less by priming the running total with a series of samples, the exact number of which is determined by the standard deviation of the noise. Then we execute the following function until either it runs a set number of times, or succeeds. Then the program compares the reconstructed signal after the iteration with the ideal signals, and returns if there is a reasonable match.

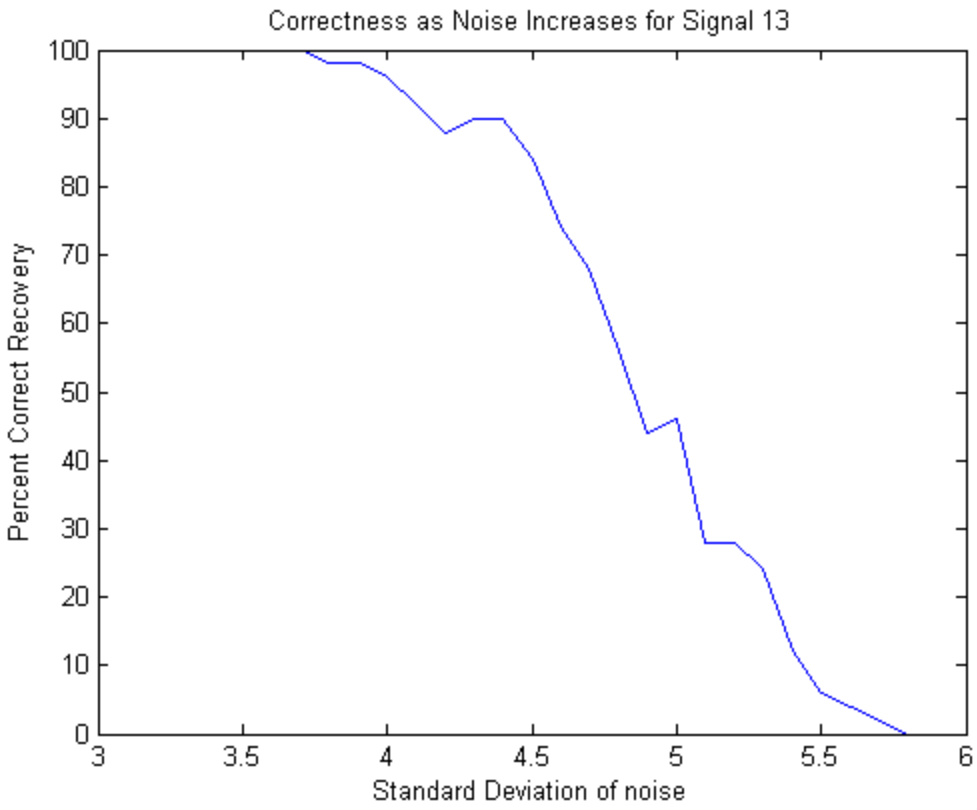
This function does the following four times: it samples the signal, updates the running total, and counts whether the maximum imaginary or real part of the Fourier transform of the signal is greater than the threshold. If at least two of the four cycles result in a value greater than the threshold, the mask's value at that point is set to the previous value of the mask; otherwise, the mask's value is set to zero. This allows a degree of leniency (which is useful in an inherently probabilistic method) and drastically reduces the probability of failure, albeit at the expense of increasing processing time.

The priming of the noise either reduces it to a standard deviation of two and a half, or three, based on the standard deviation of the noise. The method which is selected will result in fewer net samples required than the

alternative method: although processing the information can take large quantities of samples, pre-processing requires sd^2/(2.5^2) or sd^2/(3^2) samples; hence, a larger denominator can cut off incredible quantities of samples. The reason why these values in particular were selected is that experimentally we determined that at 2.5 standard deviations, most signals required only one additional sample to become fully reconstructed, while at 3 standard deviations, they took a couple dozen to a few hundred, and a maximum of a few thousand additional samples, but it appeared to terminate the vast majority of the time as shown in Figure 1 and Figure 2.



The success of a simple signal as a function of the standard deviation of the noise for a single cycle of a sinusoid with no priming(50 repetitions at each point).



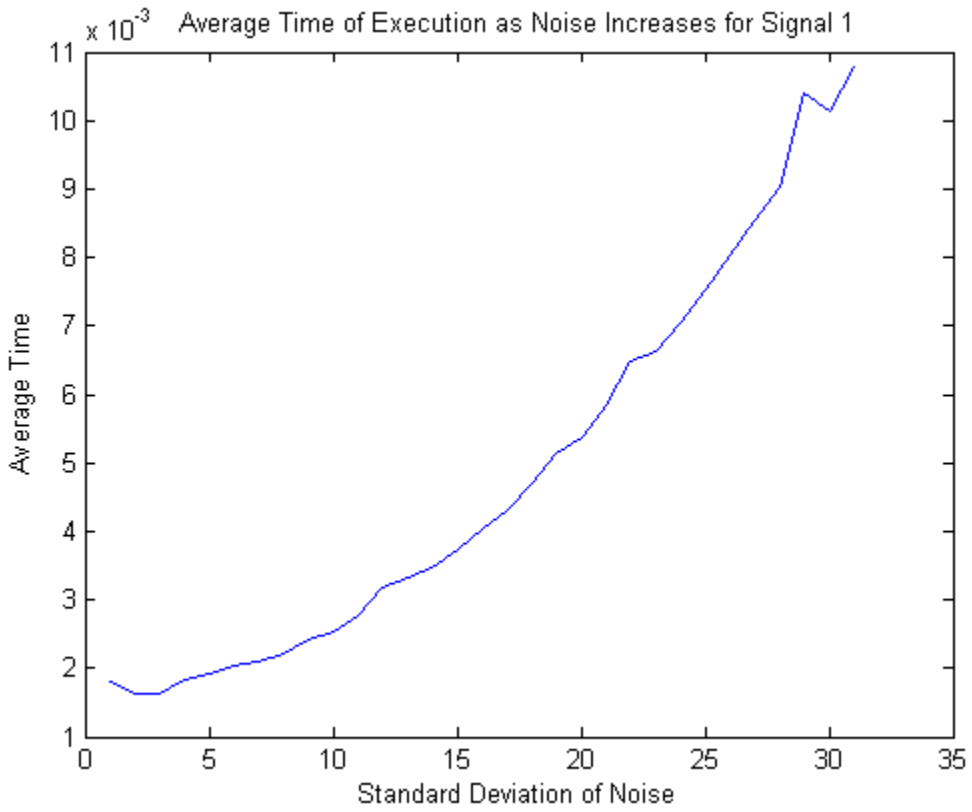
The success of a sum of a frequency one sine wave, a frequency four sine wave, and a frequency 20 cosine wave as a function of the standard deviation of the noise with no priming(50 repetitions at each point).

The priming is a necessity, since without it, the probability of success decreases rapidly as the standard deviation increases, but with it the probability of success, while not one, remains constantly above 99%. This is because after priming the effective standard deviation is always three or less, making the original signal consistently recoverable.

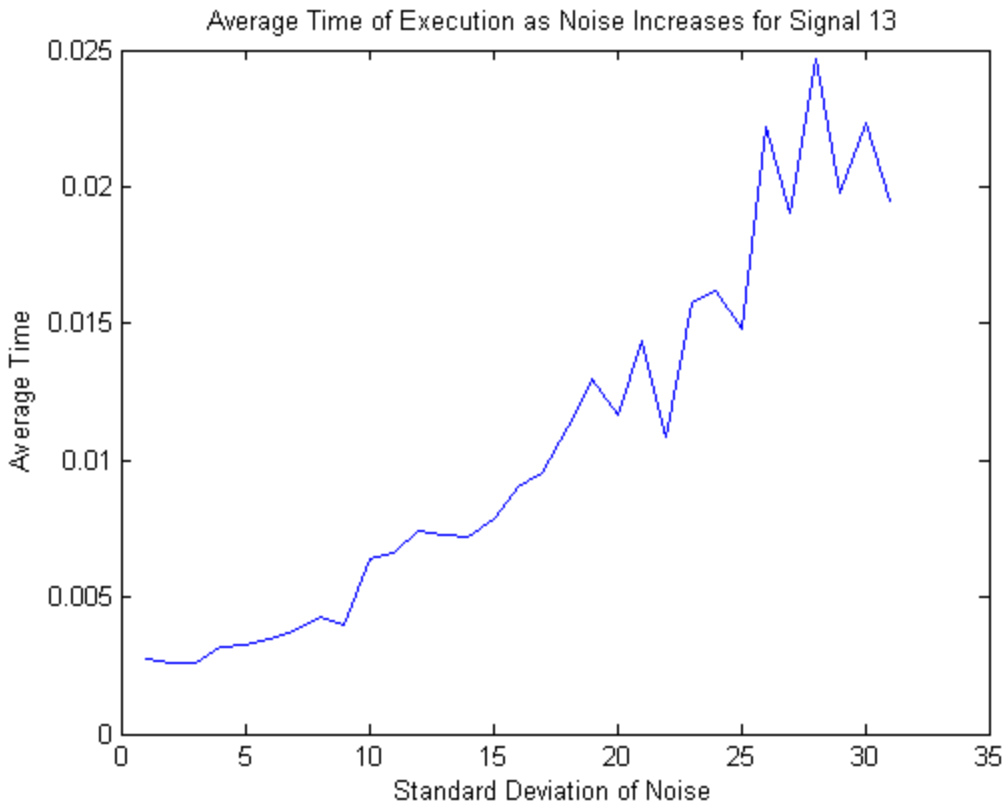
Timing Analysis

All of the initializitions before the priming, assignations etc. require O(1) operations. The priming itself requires O(sd^2) operations. The post priming processing, requires O(library size) operations since although the fft is O(N*logN), N is bounded and therefore even if the sd wasn't bounded to minimize the computation it still would always be less than a constant value. The only variable processing part is the comparisons with the library,

which is O(libsize*N*logN) with N being bounded, simplifying to O(libsize). This means that the algorithm itself is O(sd^2)+O(libsize)=O(sd^2) since libsize will tend to be small. This is supported by experimental results as shown in Figure 1 and Figure 2.

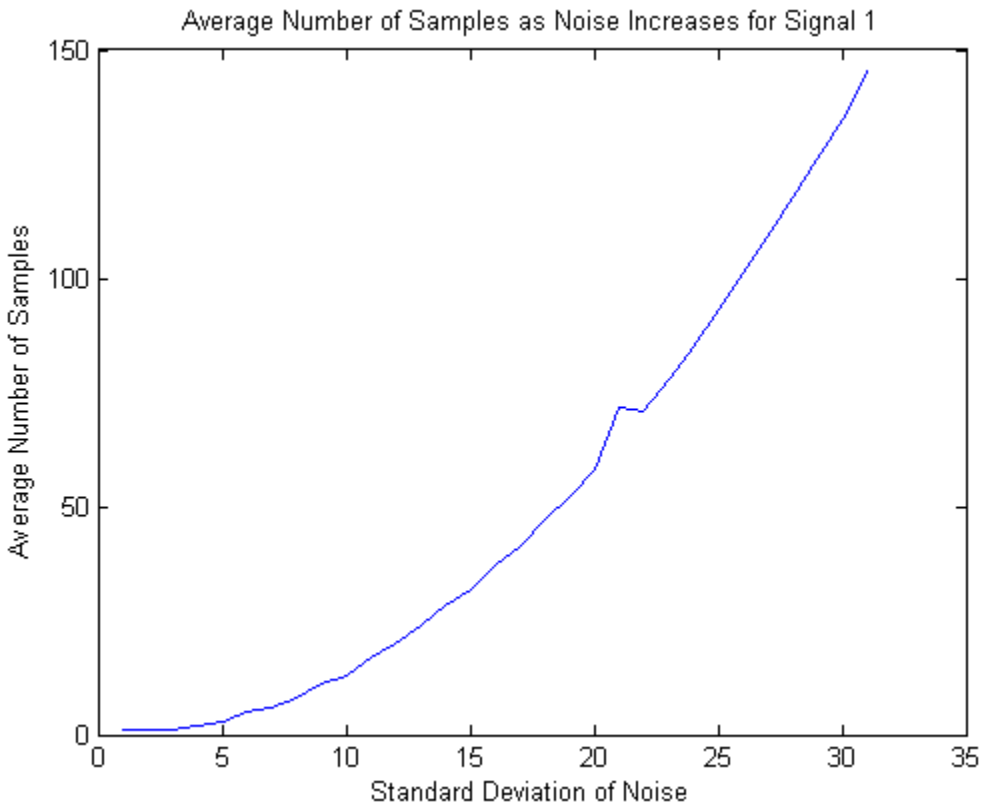


The average time of execution as a function of the standard deviation of the noise for a single cycle sinusoid (50 repetitions per point).

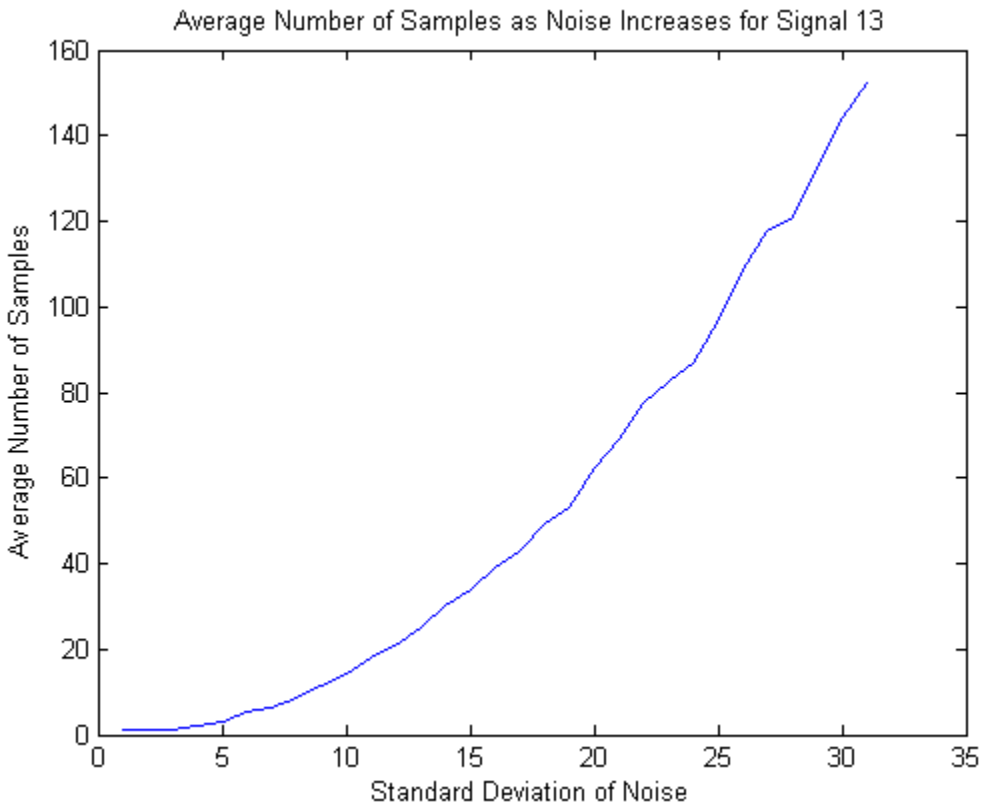


The average time of execution as a function of the standard deviation of the noise for the sum of a frequency 1, sine wave, a frequency 4 sine wave, and a frequency 20 cosine wave (50 repetitions per point).

For a real world application, it is also important to know not only the time it takes to execute the entire program, but also the number of samples that are required in order to reconstruct the signal. This is because the signal will only be transmitted for a limited time, so the hardware must be able to take the right number of samples in that time. From the processing perspective, even with very slow hardware the signal could eventually be reconstructed, but the same is not the case if the correct number of samples cannot be garnered. The number of samples required is always on O(sd^2) specifically sd^2/6.25+1 if sd<76, and sd^2/9+~200 if sd>=76. The experimental values are shown in Figure 3 and Figure 4.



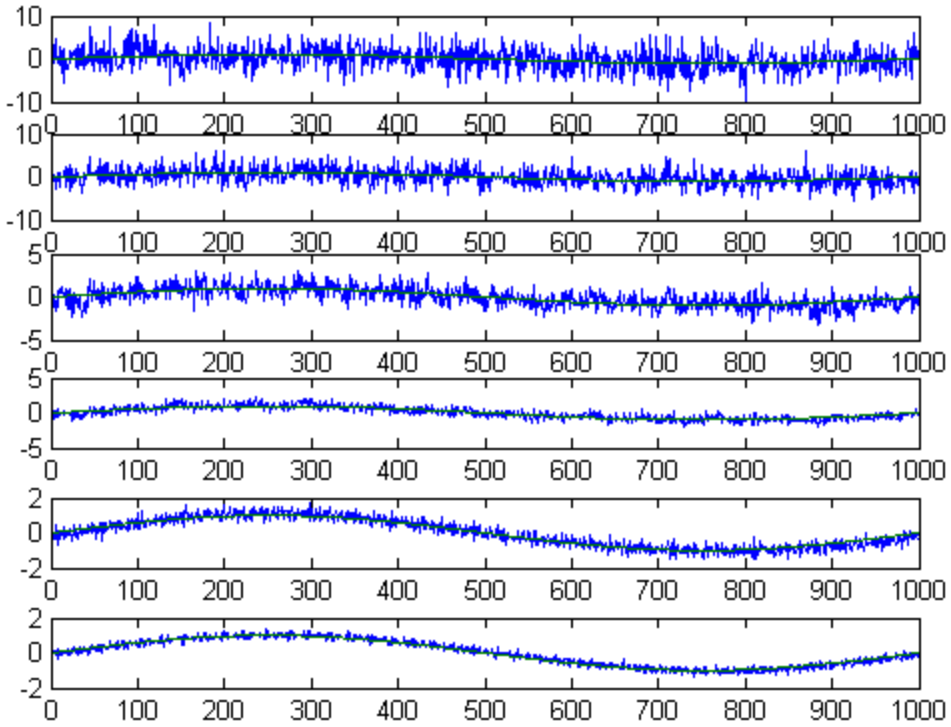
The average number of samples required to reconstruct the signal as standard deviation increases for a single cycle sine wave (50 repetitions at each point).



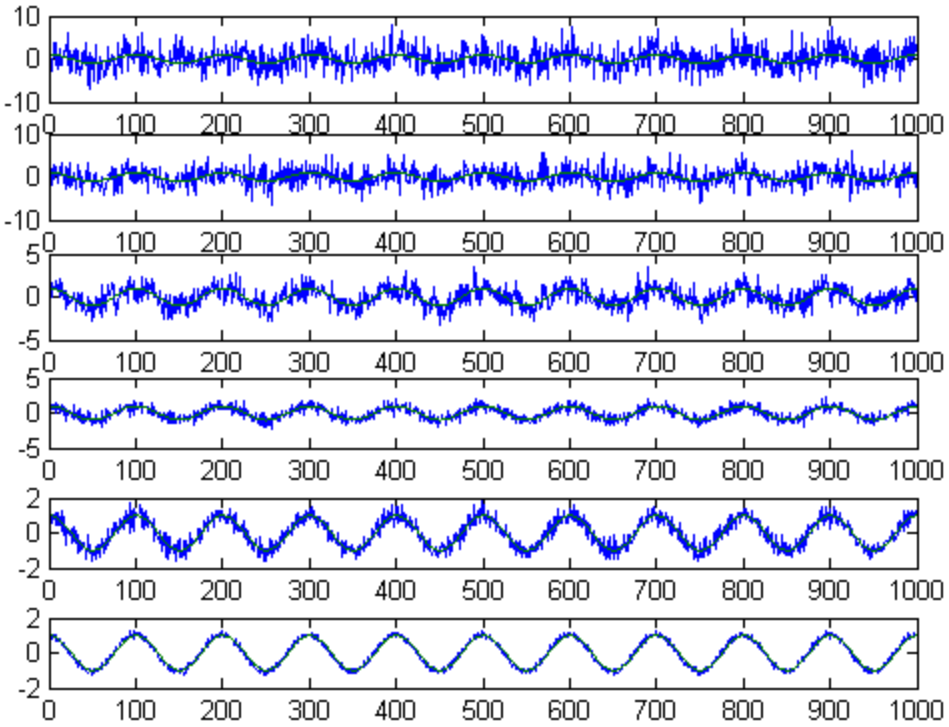
The average number of samples to reconstruct the signal as a function of the standard deviation of the noise for the sum of a frequency 1 sine wave, a frequency 4 sine wave, and a frequency 20 cosine wave (50 repetitions per point).

Examples of Execution

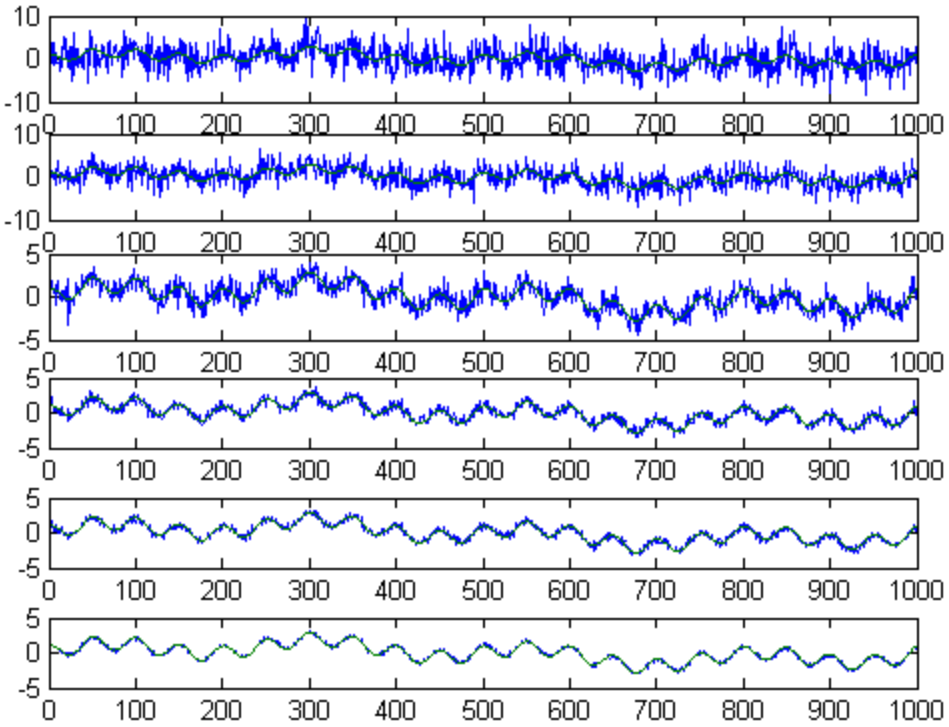
All of the following figures are comprised of a series of graphs. The first graph is the initial signal with noise, as well as the initial signal; the next four graphs are the reconstructed signal at consecutive iterations along with the initial signal; the final graph is the final reconstructed signal along with the initial signal.



Execution from algorithm for signal 1, a frequency 1 sine wave with noise standard deviation of 5.



Execution from algorithm for signal 10, a frequency 10 cosine wave with noise standard deviation of 5.



Execution from algorithm for signal 13, the sum of a frequency 1 sine wave, a frequency 4 sine wave, and a frequency 20 cosine wave with noise standard deviation of 5.

Conclusion
The Conclusion module of the Sparse Signal Recovery in the Presence of Noise collection.

Conclusion

With priming, the regression line correlating number of samples and standard deviation is roughly (sd^2)/6.5 + 1, which amounts to **O(N^2)** complexity. However, although this may seem slow, our method permits perfect recovery of complicated (albeit sparse) signals *with* **arbitrary levels of AWGN**. Thus, for reasonable data set sizes and values of standard deviation, the algorithm functions quite nicely for accurate signal reconstruction. Although if given infinite samples and time, it can recover any signal that is relatively sparse given infinite time.

Without priming the noise, and taking 50 samples, we can achieve **O(1)** complexity – extremely desirable, but the recovery percentage falls off rapidly with increase in noise standard deviation starting at standard deviation values around 3.5. Thus, this version of the algorithm could be desirable in non-critical applications where the strength of the noise is known to be low relative to that of the signal. We certainly would not recommend using the non-primed algorithm in data sensitive digital applications.

Further Avenues of Inquiry

Colored Noise

It would be interesting to extend the algorithm to accept **different types of noise** – pink, brown, purple, etc. Logically the exact same algorithm would work on these, but it would be nice to verify this experimentally.

Physical Prototype

A physical prototype of this system would allow a far better testing of the theory than simple simulation. Unfortunately, due to the cost of even moderate quality receivers and FPGAs, this was not feasible.

Dynamic Noise

One major benefit of a noise resistant system is ECCM, Electronic Counter Countermeasures (counter-jamming). It would be interesting to test whether a system using the described algorithm could resist noise from a transmitter moving towards the system (without simply taking a conservative estimate of worst-case noise during the signal reconstruction period).

Dynamic Priming

A useful addition to this algorithm, would be to be able to pick the optimal bound for priming to minimize the number of samples rather than simply choosing the best of two options.

Code
The MATLAB source code for the Sparse Signal Recovery in the Presence of Noise collection.

Code

The following is the MATLAB source code for each of the components of our project.

addNoise.m

```
function out = addNoise(sig,mean,sd,Plot)
%addNoise
%adds noise with given mean and sd to the signal
    rand=randn(1,1000)*sd+mean;
    out=sig+rand;
    if(Plot==1)
        plot(1:1000,out,1:1000,sig);
    end
end
```

sample.m

```
function out = sample(sd,plot,sig)
%sample
%samples a manually constructed signal, and adds
gaussian noise to it
%with a standard deviation that is provided
out=fft(addNoise(sig,0,sd,plot));
end
```

init.m

```
function [out,samp]=init(sig,sd)
    %averages the signal and the noise over a
number of samples to make the
    %noise level manageable
    out=sig+randn(1,1000).*sd;
    %optimize number of samples
    if sd<76
        val=6.25;
    else
        val=9;
    end
    samp=floor((ceil(sd))^2/(val));
    for n=2:samp
        out=(out.*(n-1)+sig+randn(1,1000).*sd)/n;
    end
    out=fft(out);
end
```

simpleIterate.m

```
function [mask, NSig,runT] =
simpleIterate(sigMask,threshold,run,n,sd,sig)
%simpleIterate(sigMask,threshold,run,n)
%computes an iteration of the thresholding, with a
running average of run,
%on iteration n, with the current signal mask of
sigMask
%returns the new signal mask, the current
signal(non-masked) NSig and the
%running average of the signal runT
    siz=size(sigMask);
    temp=zeros(1,siz(2));
    for i=1:4
        NSig=sample(sd,0,sig).*sigMask;
```

```
        if(n==1)
            runT=NSig;
        else
            runT=(run.*(n-1)+NSig)/n;
        end
        %temp=temp+
(max(abs(real(NSig)),abs(imag(NSig)))>threshold);
        temp=temp+
(max(abs(real(runT)),abs(imag(runT)))>threshold);
        %temp=temp+(abs(NSig)>threshold);
    end
    mask=zeros(1,siz(2));
    for l=1:siz(2)
        if(temp(l)<2)
            mask(l)=0;
        else
            mask(l)=sigMask(l);
        end
    end
end
```

testArbitary.m

```
function [flag,samples,time]=testArbitrary(sig,sd)
%Simulates the transmission of a signal in the
library, and tests whether
%or not it can be recovered.
    siglib=cat(1,sin(0:pi/500:(1000*pi-
1)/500),sin(0:pi/250:(2000*pi-
1)/500),sin(0:pi/125:(4000*pi-1)/500),sin(0:pi/50:
(10000*pi-1)/500),sin(0:pi/25:(20000*pi-1)/500));
    siglib=cat(1,siglib,sin(0:pi/500:(1000*pi-
1)/500)+sin(0:pi/50:(10000*pi-
1)/500),cos(0:pi/500:(1000*pi-
```

```
1)/500),cos(0:pi/250:(2000*pi-
1)/500),cos(0:pi/125:(4000*pi-1)/500),cos(0:pi/50:
(10000*pi-1)/500));
    siglib=cat(1,siglib,cos(0:pi/25:(20000*pi-
1)/500),cos(0:pi/25:(20000*pi-
1)/500)+sin(0:pi/500:(1000*pi-
1)/500),sin(0:pi/500:(1000*pi-
1)/500)+sin(0:pi/125:(4000*pi-1)/500)+cos(0:pi/25:
(20000*pi-1)/500));
    sigmax=max(abs(fft(siglib(sig,:))));
    threshhold=sigmax-
3*max(abs(real(fft(randn(1,1000)))));
    tolerance=.5;
    A=ones(1,1000);
    flag=0;
    tic
    [C,samples]=init(siglib(sig,:),sd);
    for i=1:10000

[A,B,C]=simpleIterate(A,threshhold,C,i+samples,sd,
siglib(sig,:));
        for j=1:size(siglib)
            if(abs(ifft(A.*C)-siglib(j,:))
<tolerance)
                flag=j;
                break;
            end
        end
        if(flag>0)
            break;
        end
    end
    samples=samples+i;
    time=toc;
end
```

Controller.m

```
function accepted = Controller(enteredpassword,sd)
%Tests whether or not a transmission of a password
will activate the system
%This simulates the noise and processing as well
as the values
    actualpassword=cat(1,13,5,10,4,2,8);
    accepted=1;
    redundancy=3;
    for i=1:size(actualpassword);
        flag=0;
        %while flag==0
        for j=1:redundancy

[flag,runs]=testArbitrary(enteredpassword(i),sd);
        end
        if(flag~=actualpassword(i))
            accepted=-i;
            break;
        end
    end
end
```

Controller2.m

```
function Controller2()
%Helper function used to graph trends
    sig=1;
    for sd=0:30
        passed=0;
        for reps=1:50
            if(testArbitrary(sig,3+sd/10)==sig)
                passed=passed+1;
            end
```

```
        end
        temp(sd*10-29)=passed
    end
    subplot(1,1,1);
    plot(temp);
end
```

References and Acknowledgements
References and Acknowledgements for the Sparse Signal Recovery in the Presence of Noise collection.

References and Acknowledgements

We wish to thank our mentor, JP Slavinsky. Without your guidance and support, this project would not have been possible.

Many thanks also to Dr. Rich Baraniuk – for creating Connexions, and (arguably more importantly), for presenting inspiring signal processing lectures throughout the semester.

J D Haupt's presentation to the Electrical Engineering department on sparse signal reconstruction was integral to the development of our algorithm, although it concentrated on efficiency rather than arbitrary reconstruction.

The algorithms and techniques discussed in the following papers were used for comparison against our algorithm:

Haupt, J. and R. Nowak. "Signal Reconstruction From Noisy Random Projections." *IEEE Trans. Info. Theory*, vol.52, no.9, pp.4036-4048, 2006. http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1683924&isnumber=35459

J. Tropp and A. Gilbert, "Signal Recovery from Partial Information via Orthogonal Matching Pursuit." *IEEE Trans. Info. Theory*, vol. 53, no. 12, pp. 4655--4666, 2007. http://www.math.lsa.umich.edu/~annacg/papers/TG05-signal-recovery-rev-v2.pdf

Team
Information about the team that created the Sparse Signal Recovery in the Presence of Noise collection.

Team

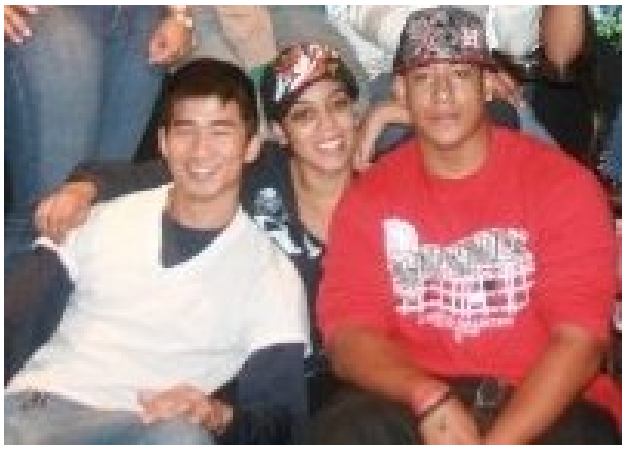
Grant Cathcart

Grant Cathcart was born in Cleveland Ohio, in 1989. Grant is currently a junior Electrical Engineering major at Rice University specializing in signals and systems. Grant is employed by the Navy as part of the Tactical Electronic Warfare Division. When not working on projects and cursing matlab, Grant likes to play chess and video games.

Graham de Wit

Graham was born in Cincinnati, OH in 1989. However, he spent most of his life in Memphis, TN, the Home of the Blues. Graham is currently a junior Electrical Engineering major at Rice University specializing in Computer Engineering. Graham enjoys convolving signals, computing Fourier Transforms, and inducing capacitor explosions. When he is not buried in problem sets, Graham spends time eating, sleeping, hanging out with friends, and composing music via his synthesizers.

Nicholas “Re'Sean” Newton



Nicholas Newton was born in Wichita Falls,TX in 1989. He is currently an junior Electrical Engineering major with a specialization in Computer Engineering at Rice University . He plans to attend Graduate School after he graduates from Rice. Nicholas has a deep passion for the Computer Engineering industry and computers in general. Outside of his academic career, Nicholas enjoys working out and a number of different sports.